Objective-C and Garbage Collection Advancements

And Blocks, too!

Session 419

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Objective-C and Garbage Collection Advancements

And Blocks, too!

Blaine Garst
Runtimes Wizard

Patrick Beard
Runtimes Guru

Introduction

- Core runtimes
  - Objective-C runtime
  - Garbage Collection
- C Language extensions
  - Blocks

Objective-C Runtime Advancements

Performance

Global Selector Uniquing

- Part of dyld shared cache
- Globally shared selectors and references
  - 540K (68512 unique selectors)
  - 48 pages (200,000 selector references)
- Faster launch time for all

Hybrid Vtable Dispatch

- Used for 16 most frequently used selectors
  - `-retain`, `-release`, `-autorelease`, `-length`, `-objectAtIndex:`, …
  - 70% of all message sends globally
- 30–50% faster than objc_msgSend()
  - 7 instructions vs. 28
  - Two words per unique message send
- Non-fragile vtables
  - No C++-style binary incompatibility

Hybrid Vtable Dispatch

First message send

```
double sum(NSArray *numbers) {
  double result = 0.0;
  NSUInteger i, N = [numbers length];
  for (i = 0; i < N; i++) {
    NSNumber *number = [numbers objectAtIndex:i];
    result += [number doubleValue];
  }
  return result;
}
```

Hybrid Vtable Dispatch

First message send

```
double sum(NSArray *numbers) {
  double result = 0.0;
  NSUInteger i, N = [numbers length];
  for (i = 0; i < N; i++) {
    ...
    static struct { SEL sel; IMP invoke; } message = {
      @selector(objectAtIndex:), (IMP)objc_msgSend_fixup
    };
    message.invoke(numbers, (SEL)&message, i);
    ...
  }
  return result;
}
```

Hybrid Vtable Dispatch

Next message send

```
double sum(NSArray *numbers) {
  double result = 0.0;
  NSUInteger i, N = [numbers length];
  for (i = 0; i < N; i++) {
    ...
    static struct { SEL sel; IMP invoke; } message = {
      @selector(objectAtIndex:), (IMP) objc_msgSend_vtable10
    };
    message.invoke(numbers, (SEL)&message, i);
    ...
  }
  return result;
}
```

Debugging

Specialized vtable dispatch code

```
(gdb) stepi
0x00007fff88148674 in objc_msgSend_fixup ()
(gdb) disassemble
Dump of assembler code for function objc_msgSend_fixup:
0x00007fff88148674 <objc_msgSend_fixup+0>:  test   %rdi,%rdi
0x00007fff88148677 <objc_msgSend_fixup+3>:  je     0x7fff88148761
```

- New symbols
  - objc_msgSend_fixup
  - objc_msgSend_fixedup
  - objc_msgSend_vtable0, …, objc_msgSend_vtable15
- Debug as if you were in ordinary objc_msgSend

Debugging

Old crash logs

```
Process:         WindowSound [4211]
Path:            /var/tmp/build/WindowSound.app/Contents/MacOS/WindowSound
Identifier:      com.yourcompany.WindowSound
Version:         1.0 (1)
Code Type:       X86-64 (Native)
Parent Process:  launchd [117]

Date/Time:       2009-06-07 21:16:05.980 -0700
OS Version:      Mac OS X 10.6 (10A380)
Report Version:  6

Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Codes: KERN_INVALID_ADDRESS at 0xffffffffffffffff
Crashed Thread:  0  Dispatch queue: com.apple.main-thread

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   libobjc.A.dylib                 0x00007fff8727553a objc_msgSend + 22
1   com.yourcompany.WindowSound     0x0000000100001a39 __-[WindowSoundAppDelegate startSound:]_block_
2   com.yourcompany.WindowSound     0x00000001000013ae -[WindowSoundAppDelegate speechSynthesizer:did
3   com.apple.AppKit                0x00007fff8427ff9c -[NSSpeechSynthesizer _handleSpeechDoneCallbac
4   com.apple.Foundation            0x00007fff87e39bac __NSThreadPerformPerform + 219
5   com.apple.CoreFoundation        0x00007fff84c852b1 __CFRunLoopDoSources0 + 1361
```

Debugging Support

Crash log enhancements

```
Process:         WindowSound [4211]
Path:            /var/tmp/build/WindowSound.app/Contents/MacOS/WindowSound
Identifier:      com.yourcompany.WindowSound
Version:         1.0 (1)
Code Type:       X86-64 (Native)
Parent Process:  launchd [117]

Date/Time:       2009-06-07 21:16:05.980 -0700
OS Version:      Mac OS X 10.6 (10A380)
Report Version:  6

Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Codes: KERN_INVALID_ADDRESS at 0xffffffffffffffff
Crashed Thread:  0  Dispatch queue: com.apple.main-thread

Application Specific Information:
objc_msgSend() selector name: play
objc[4211]: garbage collection is ON

Thread 0 Crashed:  Dispatch queue: com.apple.main-thread
0   libobjc.A.dylib                 0x00007fff8727553a objc_msgSend + 22
1   com.yourcompany.WindowSound     0x0000000100001a39 __-[WindowSoundAppDelegate startSound:]_block_
2   com.yourcompany.WindowSound     0x00000001000013ae -[WindowSoundAppDelegate speechSynthesizer:did
3   com.apple.AppKit                0x00007fff8427ff9c -[NSSpeechSynthesizer _handleSpeechDoneCallbac
4   com.apple.Foundation            0x00007fff87e39bac __NSThreadPerformPerform + 219
5   com.apple.CoreFoundation        0x00007fff84c852b1 __CFRunLoopDoSources0 + 1361
```

Synthesized Instance Variables

Property Synthesis

```
@interface Person : NSObject {
    NSString *name;
}
@property(copy) NSString *name;
@end

@implementation Person

@synthesize name;

-(void)dealloc {
    [name release];
    [super dealloc];
}
@end
```

Property Synthesis

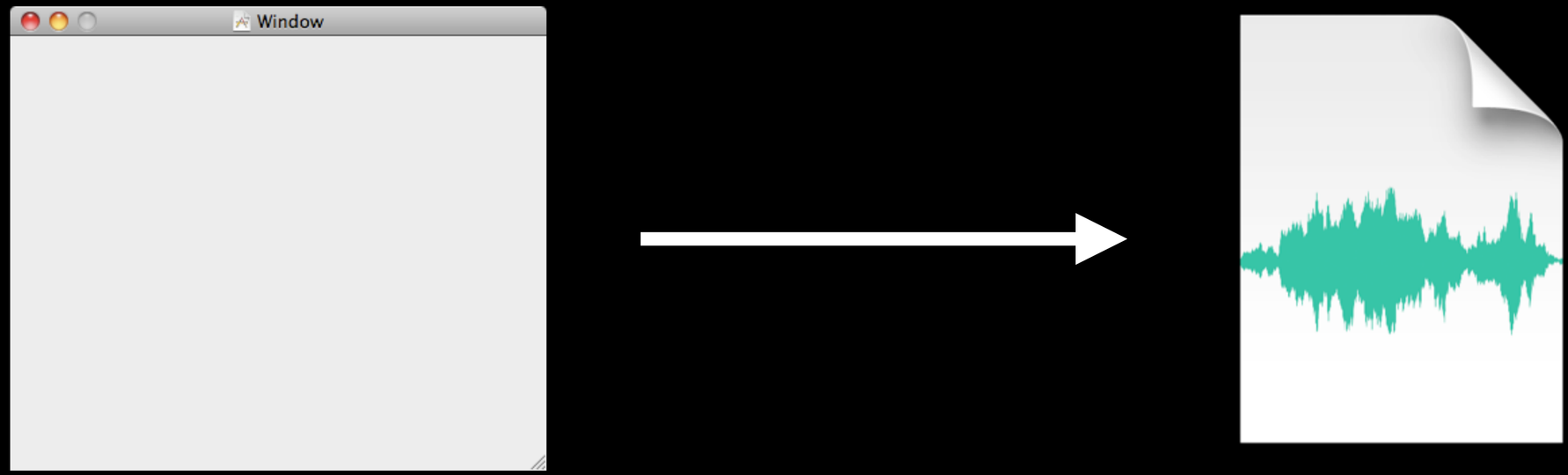
```
@interface Person : NSObject {
    // Look Ma, No Instance Variable!
}
@property(copy) NSString *name;
@end

@implementation Person

@synthesize name;

-(void)dealloc {
    [name release];
    [super dealloc];
}
@end
```

One Small Problem…

```
@interface Person : NSObject {
    // Look Ma, No Instance Variable!
}
@property(copy) NSString *name;
@end

@implementation Person

@synthesize name;

-(void)dealloc {
    [name release];
     error: 'name' undeclared (first use in this function)
    [super dealloc];
}
@end
```

Associative References

Making Associations




```
@interface NSWindow (Sounds)
@property(retain) NSSound *sound;
@end
```

Associating a Sound

```
// NSWindowDidBecomeKeyNotification notification
observer:

- (void)windowDidBecomeKey:(NSNotification *)note {
    NSWindow *window = note.object;
    if (window.sound == nil) {
        window.sound = [NSSound soundNamed:@"Ping"];
    }
    [window.sound play];
}

- (void)windowDidResignKey:(NSNotification *)note {
    NSWindow *window = note.object;
    [window.sound stop];
}
```

JavaScript or Ruby

```
myWindow["sound"] = mySound;
```

Objective-C Issues

- Storage?
  - Global, weak > strong NSMapTable
  - Subtle leaks under GC
- Lifetime?
  - Subclassing not always feasible

Using Associative References

```
@interface NSWindow (Sounds)
@property(retain) NSSound *sound;
@end

@implementation NSWindow (Sounds)
static char sound_key; // address is always unique.

- (NSSound *)sound {
  return objc_getAssociatedObject(self, &sound_key);
}

- (void)setSource:(NSSound *)sound {
  objc_setAssociatedObject(self, &sound_key, sound,
                           OBJC_ASSOCIATION_RETAIN);
}
@end
```

Associative References API

```
/* #include <objc/runtime.h> */

/* objc_setAssociatedObject() options */
enum { OBJC_ASSOCIATION_ASSIGN, OBJC_ASSOCIATION_RETAIN,
       OBJC_ASSOCIATION_COPY, ... };
typedef uintptr_t objc_AssociationPolicy;

void objc_setAssociatedObject(id object, void *key, id value,
                              objc_AssociationPolicy policy);
```

- Association policies
  - @property(assign, retain, copy)
- See sample code "AssociativeReferenceTricks" for more examples

Objective-C Runtime Advancements

Summary

- Performance
- Debugging
- Bug fixes
- Associative references

Garbage Collection Advancements

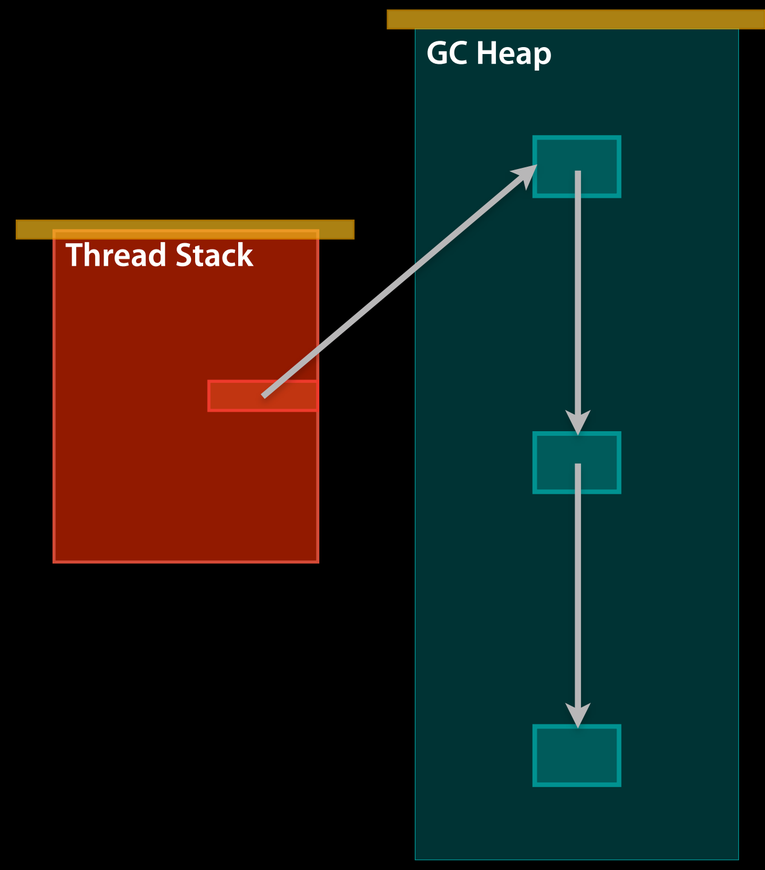
Snow Leopard and Garbage Collection

- All Cocoa/CF frameworks
- 64-bit GC system applications
  - Xcode
  - System Preferences
  - Screen Saver
  - Automator

Thread Local Collection

Thread Local Collection

- New collector mode
- Local hypothesis
- Fast
- Efficient
- Scalable



Thread Local Collection

- Scope
  - Small objects (1–96 bytes in 64-bit mode)
  - Retain count 0 ONLY
- Triggering
  - -[NSAutoreleasePool drain]
  - -[NSGarbageCollector collectIfNeeded]
  - objc_collect(OBJC_COLLECT_IF_NEEDED);
- Forcing
  - -[NGarbageCollector collectExhaustively]
  - objc_collect(OBJC_EXHAUSTIVE_COLLECTION);

Inhibiting Thread Local Collection

- Objects *escape* when:
  - Passed to -[NSGarbageCollector disableCollectorForPointer:] or CFRetain()
  - Assigned to global variable
  - Weakly or associatively referenced
  - Assigned to an instance variable of global object

Holding a Lock? Don’t Call Drain!

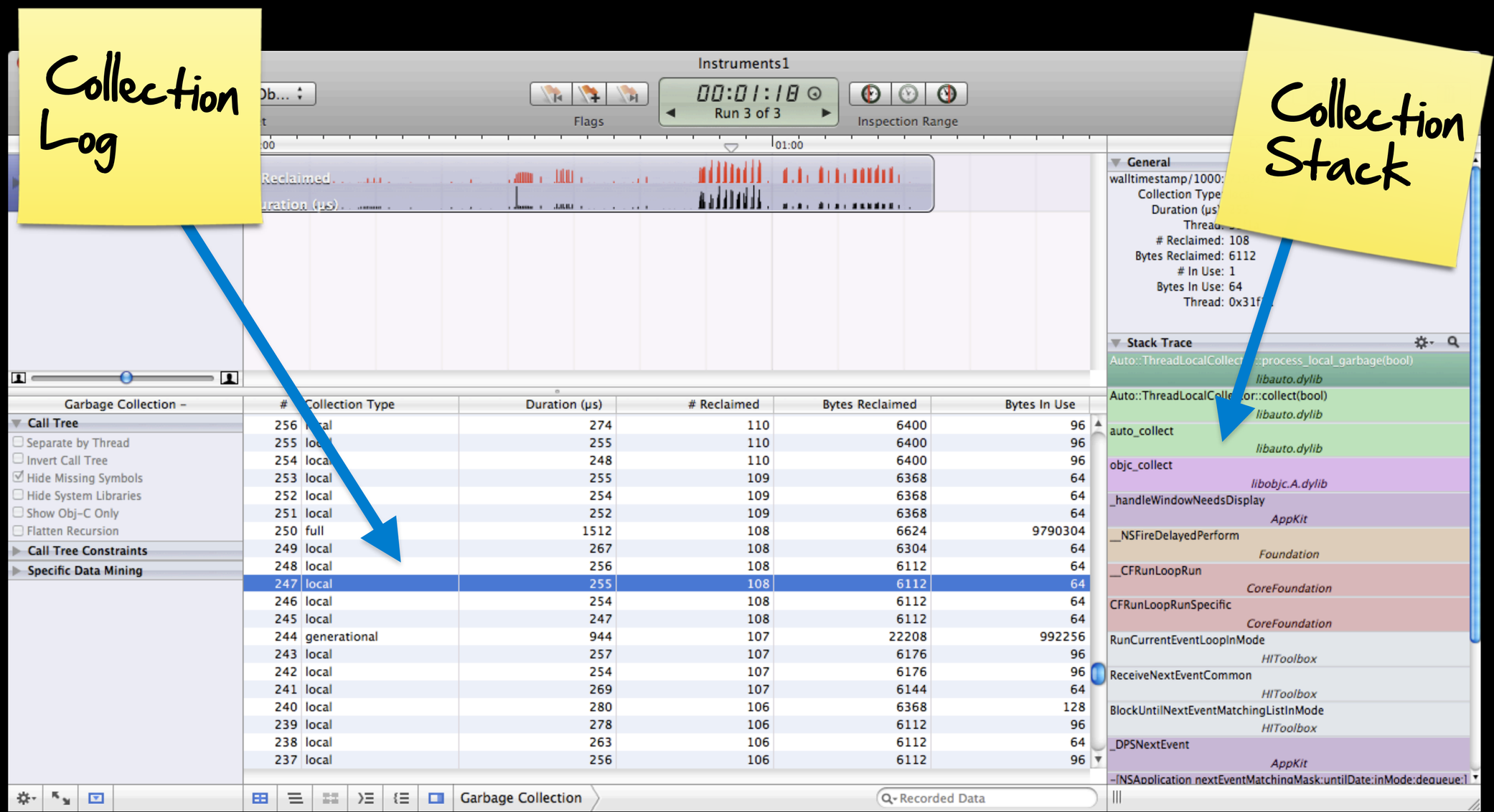
```
static NSLock *myLock;

- (void)dealloc {
    [myLock lock];
        NSLog(@"deallocating!");
    [myLock unlock];
    [super dealloc];
}

+ (void)alwaysDeadlocks {
    [myLock lock];
        NSAutoreleasePool *pool = [NSAutoreleasePool new];
        [[MyClass new] autorelease];
        [pool drain];
    [myLock unlock];
}
```

Holding a Lock? Don’t Call Collect!

```
static NSLock *myLock;

- (void)finalize {
    [myLock lock];
        NSLog(@"deallocating!");
    [myLock unlock];
    [super finalize];
}

+ (void)alwaysDeadlocks {
    [myLock lock];
        [MyClass new];
        objc_collect(OBJC_EXHAUSTIVE_COLLECTION);
    [myLock unlock];
}
```

Instruments and Garbage Collection

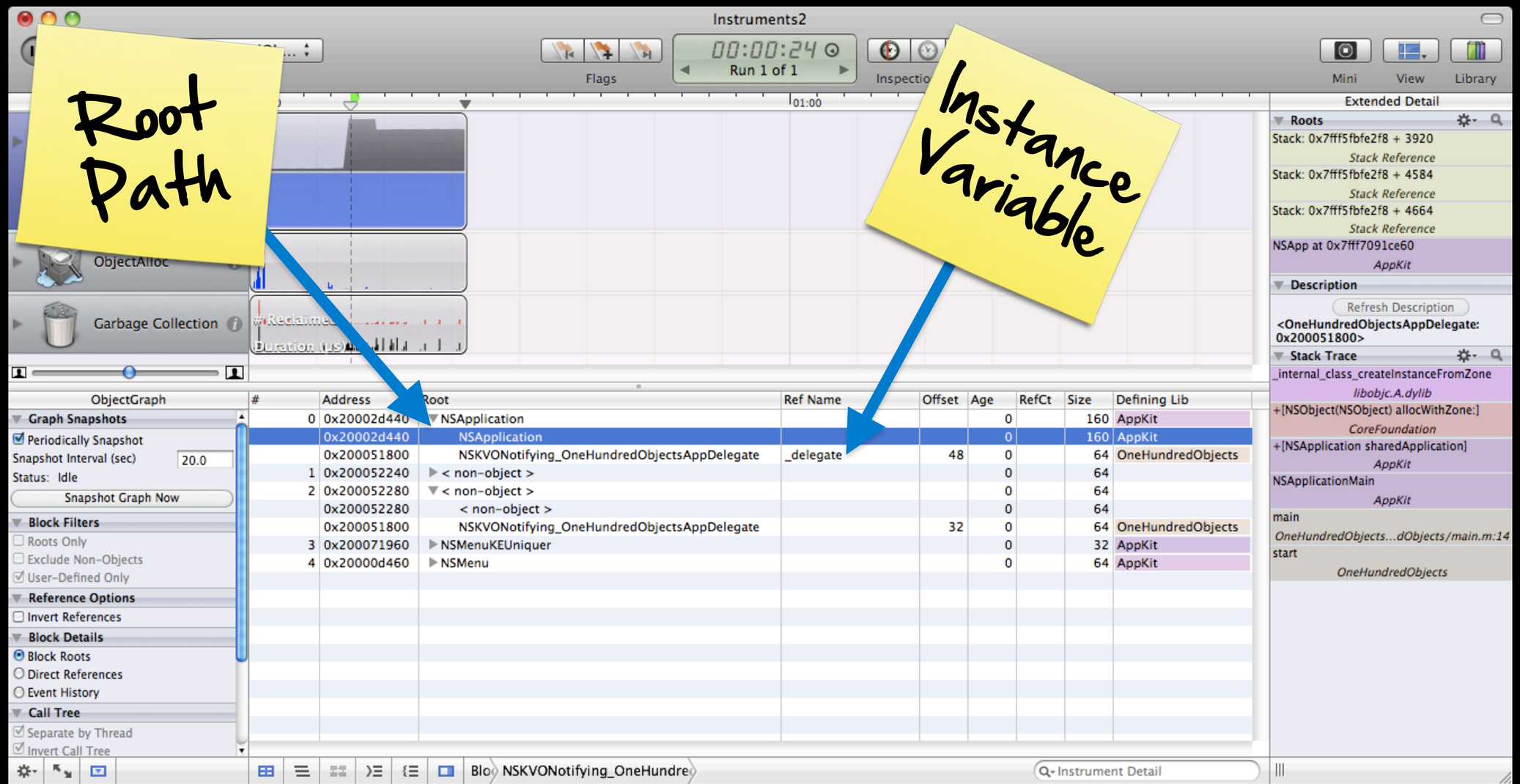
Comparing collection performance



#	Collection Type	Duration (µs)	# Reclaimed	Bytes Reclaimed	Bytes In Use
256	local	274	110	6400	96
255	local	255	110	6400	96
254	local	248	110	6400	96
253	local	255	109	6368	64
252	local	254	109	6368	64
251	local	252	109	6368	64
250	full	1512	108	6624	9790304
249	local	267	108	6304	64
248	local	256	108	6112	64
247	local	255	108	6112	64
246	local	254	108	6112	64
245	local	247	108	6112	64
244	generational	944	107	22208	992256
243	local	257	107	6176	96
242	local	254	107	6176	96
241	local	269	107	6144	64
240	local	280	106	6368	128
239	local	278	106	6112	96
238	local	263	106	6112	64
237	local	256	106	6112	96

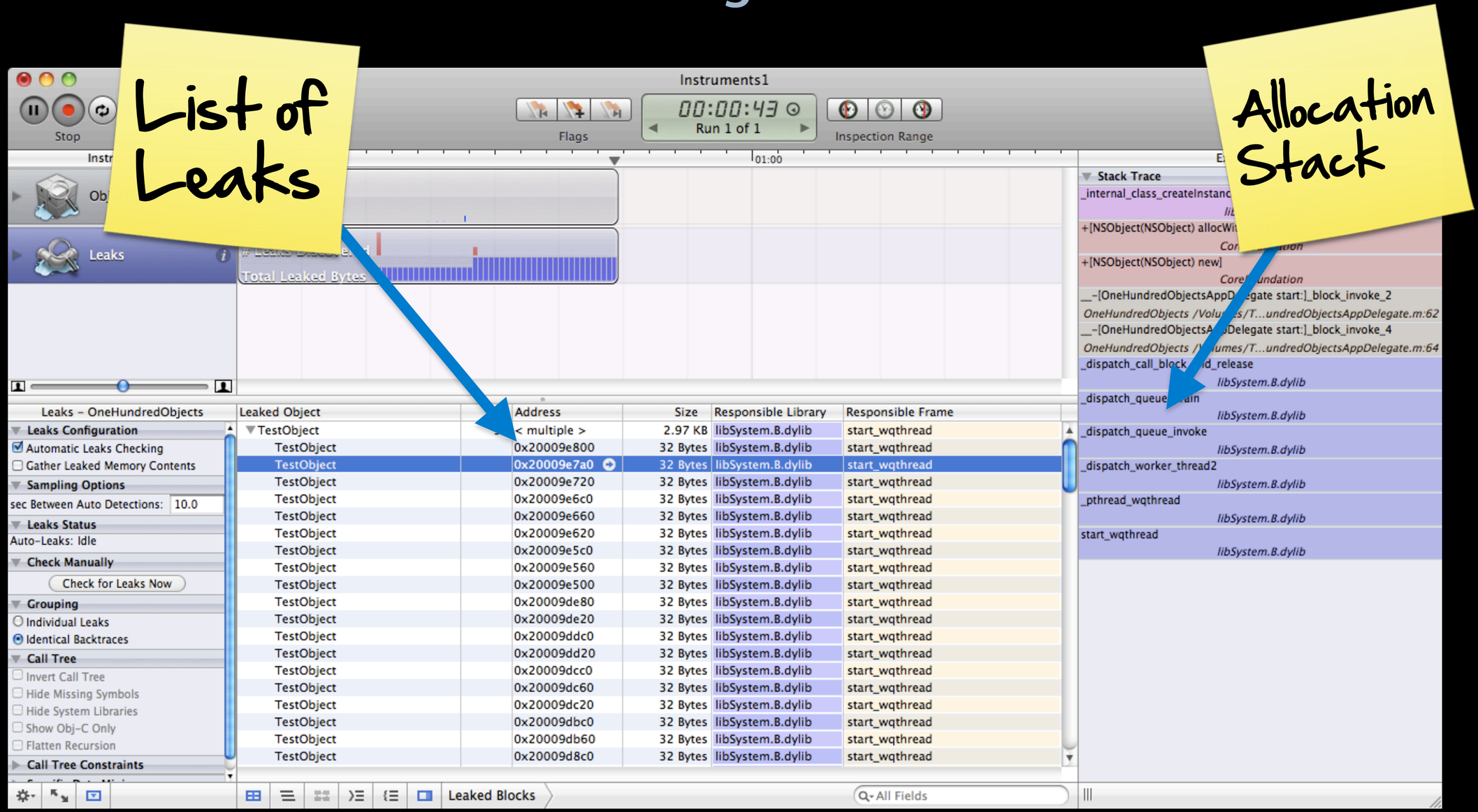
Instruments and Garbage Collection

Root paths



Instruments and Garbage Collection

Finding leaks



Related Sessions

What's New in Instruments	Pacific Heights Wednesday 3:30 PM

GDB 101

- Info commands
  - info malloc-history pointer
  - info gc-references pointer
  - info gc-roots pointer
- Handy functions
  - p (BOOL)objc_collectingEnabled()
  - p (long)CFGetRetainCount(pointer)
  - p (long)malloc_size(pointer)
  - p (BOOL)objc_is_finalized(pointer)

Environment Variables

- Guard Pages
  - AUTO_USE_GUARDS = YES
- Logging
  - AUTO_LOG_COLLECTIONS = YES
  - AUTO_REFERENCE_COUNT_LOGGING = YES
- Modes
  - AUTO_USE_TLC = NO
  - AUTO_DISABLE_GENERATIONAL = YES

Open Source

- Leopard 10.5.3 and later
  - http://opensource.apple.com/source/autozone/autozone-77.1/
  - http://opensource.apple.com/tarballs/autozone/autozone-77.1.tar.gz
- Apache License Version 2.0
- MacRuby
- C Standards Committee presentation
  - *Apple's Extensions To C*
  - http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1370.pdf

Blocks

Topics

- High-level overview
- Syntax
- Lifetime
- Objective-C specializations
- Surprises
- Status

```
(lambda (a) (add a d))
```

Scheme

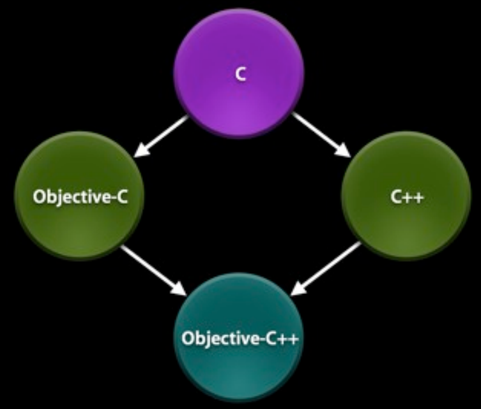
LISP

SmallTalk

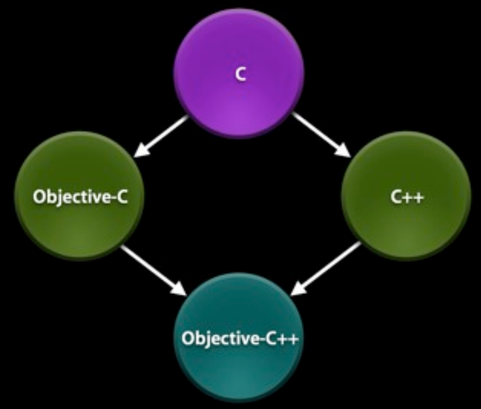
```
10 timesRepeat:[pen turn:d; draw]
```

Ruby

```
z.each {|val| puts(val + d.to_s) }
```




```
int d = i % 10;
repeat(10, ^{ putc('0'+d); });
```




```
int d = i % 10;
repeat(10, ^{ putc('0'+d); });
```

A Block Reference Declaration

```
void (^workBlk)(void);
```

Use Typedefs

```
typedef void (^workBlk_t)(void);
```

Do Block N Times Function

```
typedef void (^workBlk_t)(void);

void
repeat(int n, workBlk_t block) {
        for (int i = 0; i < n; ++i)
                block();
}
```

Why Blocks?

- Smaller, easier, more precise code
- Total of >100 uses of blocks in Snow Leopard APIs
  - Callbacks
  - Concurrency
  - Traditional “collection” uses: Iterate, map, reduce, wrap

Callbacks Using qsort_r

```
void qsort_r (
        void *array,
        size_t nElems,
        size_t width,
        void *callback_argument,
        int (*compare)(void *l, void *r, void *cb)
);
```

Callbacks Using qsort_r

1. Declare custom structure type

```
typedef struct {
    int stringOption;
    BOOL ageFirst;
    ...;
} PeopleSort_t;
```

Callbacks Using qsort_r

2. Write compare function using custom struct

```
int myCustomCompare(void *p1, void *p2,
                                  void *ctx) {
    ...
    if (((PeopleSort_t *)ctx)->ageFirst)
            ...
    ...
}
```

Callbacks Using qsort_r

3. Declare and initialize custom structure

```
void sortPeople(BOOL ageFirst) {
    int options = ...
    PeopleSort_t ctx = {
        ageFirst,
        options,
        ...};
```

Callbacks Using qsort_r

4. Call qsort_r

```
...
qsort_r(array, nelems, width,
                    &ctx, myCustomCompare);
```

New: qsort_b

```
void qsort_b (
        void *array,
        size_t nElems,
        size_t width,
        int (^compare)(void *l, void *r)
);
```

New: qsort_b

1. call qsort_b

```
void sortPeople(BOOL ageFirst) {
    int options = ...
qsort_b(array, nelems, width, ^ {
        ...
        if (ageFirst > 0)
            ...;
        ...
 });
```

Concurrency: Send Work to GCD Queue

```
int a;
int b;
...

dispatch_async(queue, ^ {     // don't wait
    ...
     ... = a + b;
});
```

Synchronization: Do Work Exactly Once!

```
static id Singleton;

static dispatch_once_t SingletonGuard;

id getSingleton() {
    ...
    dispatch_once(&SingletonGuard, ^ {
        Singleton = [[FooBar alloc] init];
    });
    return Singleton;
}
```

What Is a Block?

- Data structure that
  - References a function expression that you write
  - Acts like a function when invoked
  - Holds const versions of stack variables that you use in that code
  - References shared __block variables that you can mutate
- Lifetime
  - Starts on the stack
  - Can be copied to the heap

Mutating __block Variables

```
id findKeyForValue(NSDictionary *dict, id sought) {
    __block id result = nil;
    [dict enumerateKeysAndObjectsUsingBlock:
        ^ (id key, id object, BOOL *stop) {
            if ([object isEqual:sought)]) {
                result = key;
                *stop = YES;
            }
        }];
    return result;
}
```

```
id findKeyForValue(NSDictionary *dict, id sought) {




}
```

```
id findKeyForValue(NSDictionary *dict, id sought) {
    __block id result = nil;




    return result;
}
```

```
id findKeyForValue(NSDictionary *dict, id sought) {
    __block id result = nil;
    [dict enumerateKeysAndObjectsUsingBlock:



        ];
    return result;
}
```

```
id findKeyForValue(NSDictionary *dict, id sought) {
    __block id result = nil;
    [dict enumerateKeysAndObjectsUsingBlock:
        ^ (id key, id object, BOOL *stop) {



        }];
    return result;
}
```

```
id findKeyForValue(NSDictionary *dict, id sought) {
    __block id result = nil;
    [dict enumerateKeysAndObjectsUsingBlock:
        ^ (id key, id object, BOOL *stop) {
            if ([object isEqual:sought)]) {


            }
        }];
    return result;
}
```

```
id findKeyForValue(NSDictionary *dict, id sought) {
    __block id result = nil;
    [dict enumerateKeysAndObjectsUsingBlock:
        ^ (id key, id object, BOOL *stop) {
            if ([object isEqual:sought)]) {
                result = key;
                *stop = YES;
            }
        }];
    return result;
}
```

```
id findKeyForValue(NSDictionary *dict, id sought) {
    __block id result = nil;
    [dict enumerateKeysAndObjectsUsingBlock:
        ^ (id key, id object, BOOL *stop) {
            if ([object isEqual:sought)]) {
                result = key;
                *stop = YES;
            }
        }];
    return result;
}
```

__block Variables

- Allowed within function or inside Block object code
- Mutable
- Shared with originating scope and any Blocks that use them
- Starts on stack, copied to heap if any Block using it is copied
- Restrictions
  - No variable length arrays (gnu extension)
  - No structures with C99 variable length arrays
  - Should not take & of a __block variable (it can change!)

Recap: Block Literal Expression Syntax

- A Block literal that takes no parameters; return type inferred
  - ^ { *statements* }
  - ^ { printf("hello world\n"); } // returns void
  - ^ { return 42; } // returns int
- A Block literal taking parameters; return type inferred
  - ^ (*parameter-list*) { *statements* }
  - ^ (int parameter) { printf("%d was passed in\n", parameter); }
- Full syntax includes a declared return type
  - ^ *return-type* (*parameter-list*) { *statements* }
  - ^ char (char c) { if (c > '0') return c; else return 0; }

Return Type Inferencing Subtlety #1

Problem

```
^ { return 'c';  };   // returns int!!!
```

Solution #1

```
^ { return (char)'c';  };
```

Solution #2

```
^ char (void) { return 'c';  };
```

Return Type Inferencing Subtlety #2

Problem

```
^ { return a == b;  };   // returns int!!!
```

Solution #1

```
^ { return (BOOL)(a == b);  };
```

Solution #2

```
^ BOOL (void) { return a == b;  };
```

Return Type Inferencing Subtlety #3

```
enum { TEN = 10L; };    // int initialized with long

^ { return TEN;  };   // returns int!!!
```

```
sortedArray = [array sortUsingComparator:
    ^ (id o1, id o2) {
          ....
              return NSOrderedAscending;
          ....
              return NSOrderedDescending;
          ....
              return NSOrderedSame;
    }];
          /*  COMPILE ERROR !!!!  */
```

Return Type Inferencing Subtlety #3 Solution

```
enum { TEN = 10L; };    // int initialized with long

^ { return (long)TEN;  };   // returns long
```

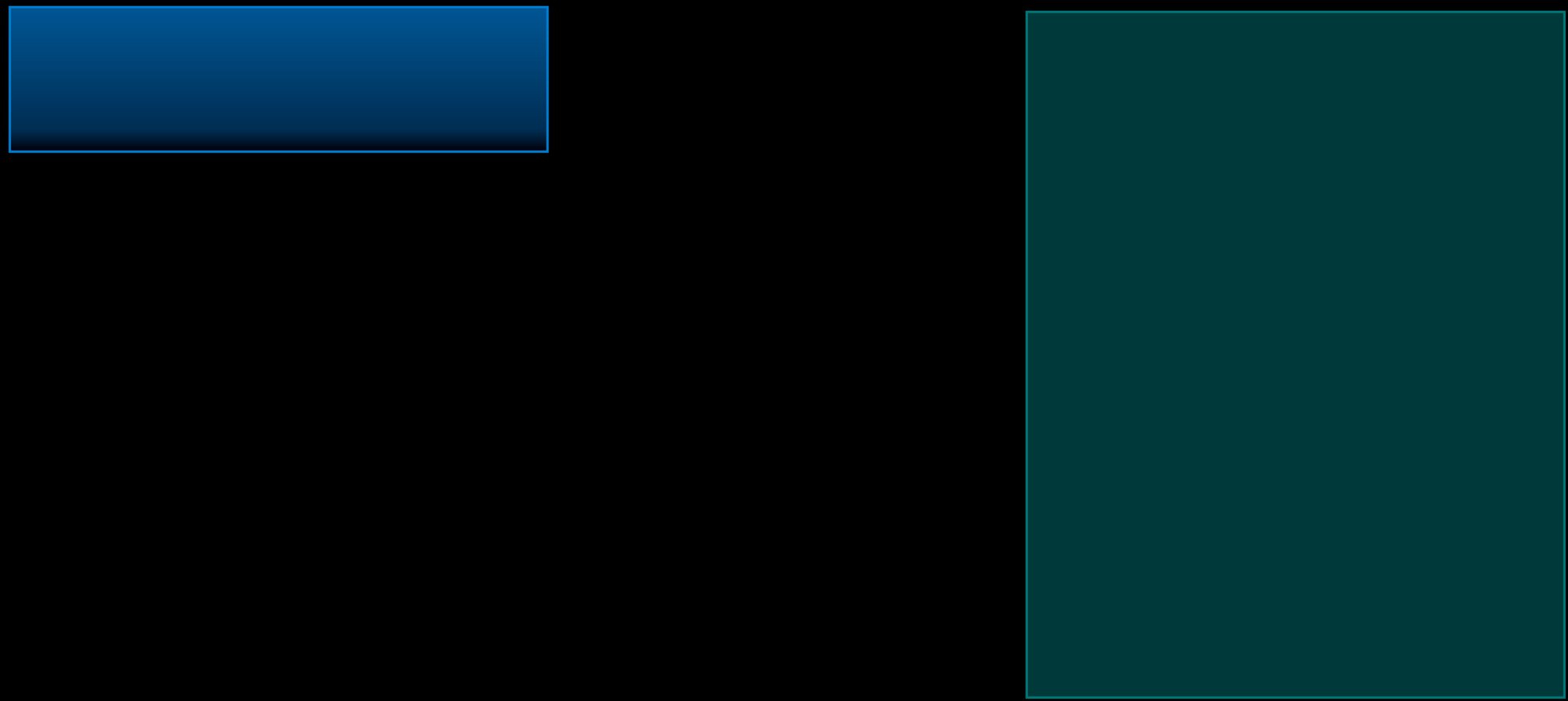
```
sortedArray = [array sortUsingComparator:
    ^ NSComparisonResult (id o1, id o2) {
          ....
              return NSOrderedAscending;
          ....
              return NSOrderedDescending;
          ....
              return NSOrderedSame;
    }];
```

Block Declarator Syntax

- Like function pointers, except ^ instead of *
  - void (^aBlock)(int);
- Objective-C arguments use abstract declarator form
  - - (void) foo:(int)x withBlock:(void (^)(int)) aBlock;
- Arrays
  - void (^blockArray[10])(int);
- Block returning function pointer, returning a Block, …
  - It is possible
  - Use typedefs

Block Lifetime

Initial State Before Function Call



Stack

Heap

State Before Copy to Heap

```
int local;

__block int shared;

const int _local;
^ {
    shared += _local;
}
```

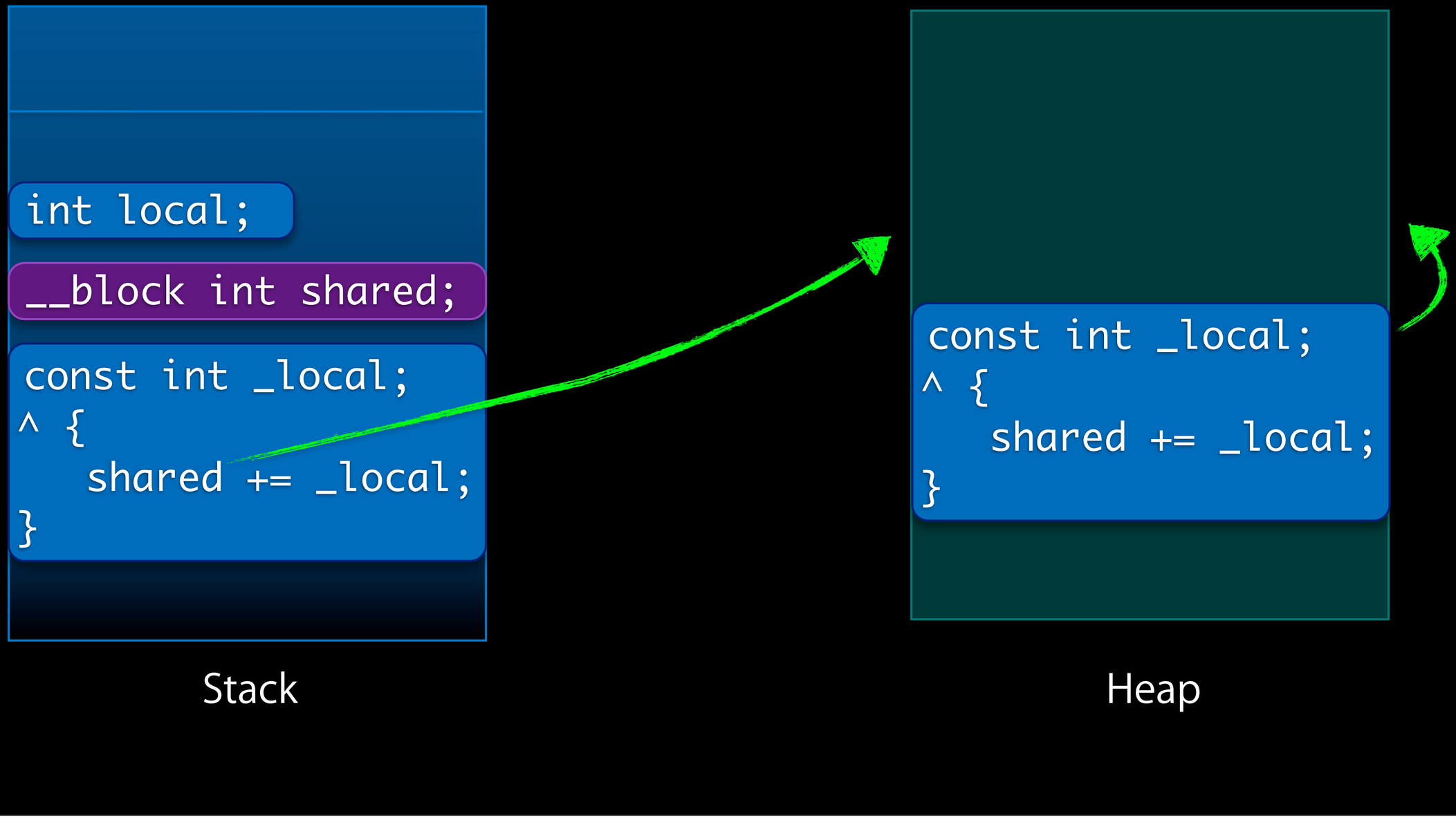
CHALK????
copied form 406...

D.

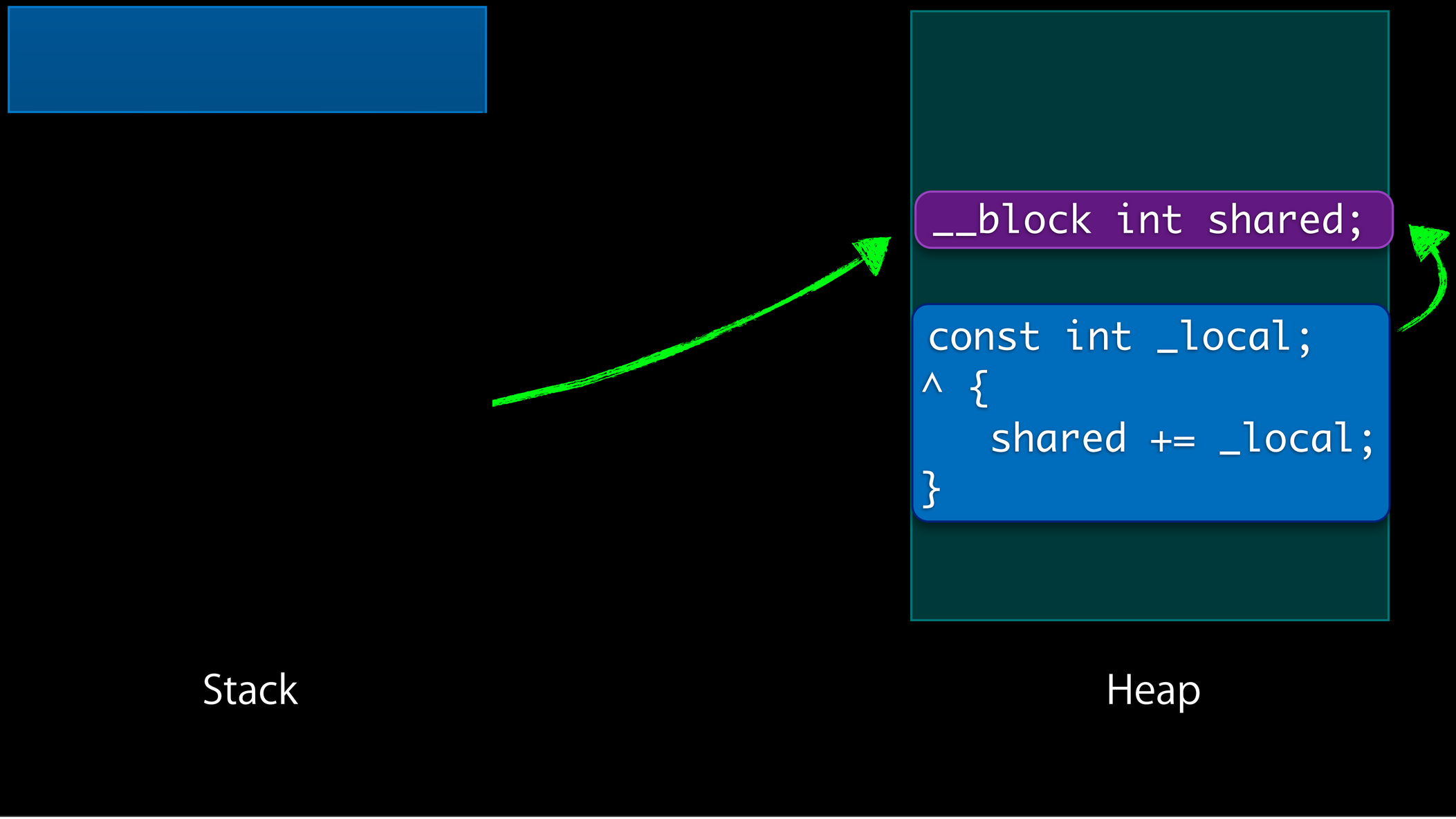
Stack

Heap

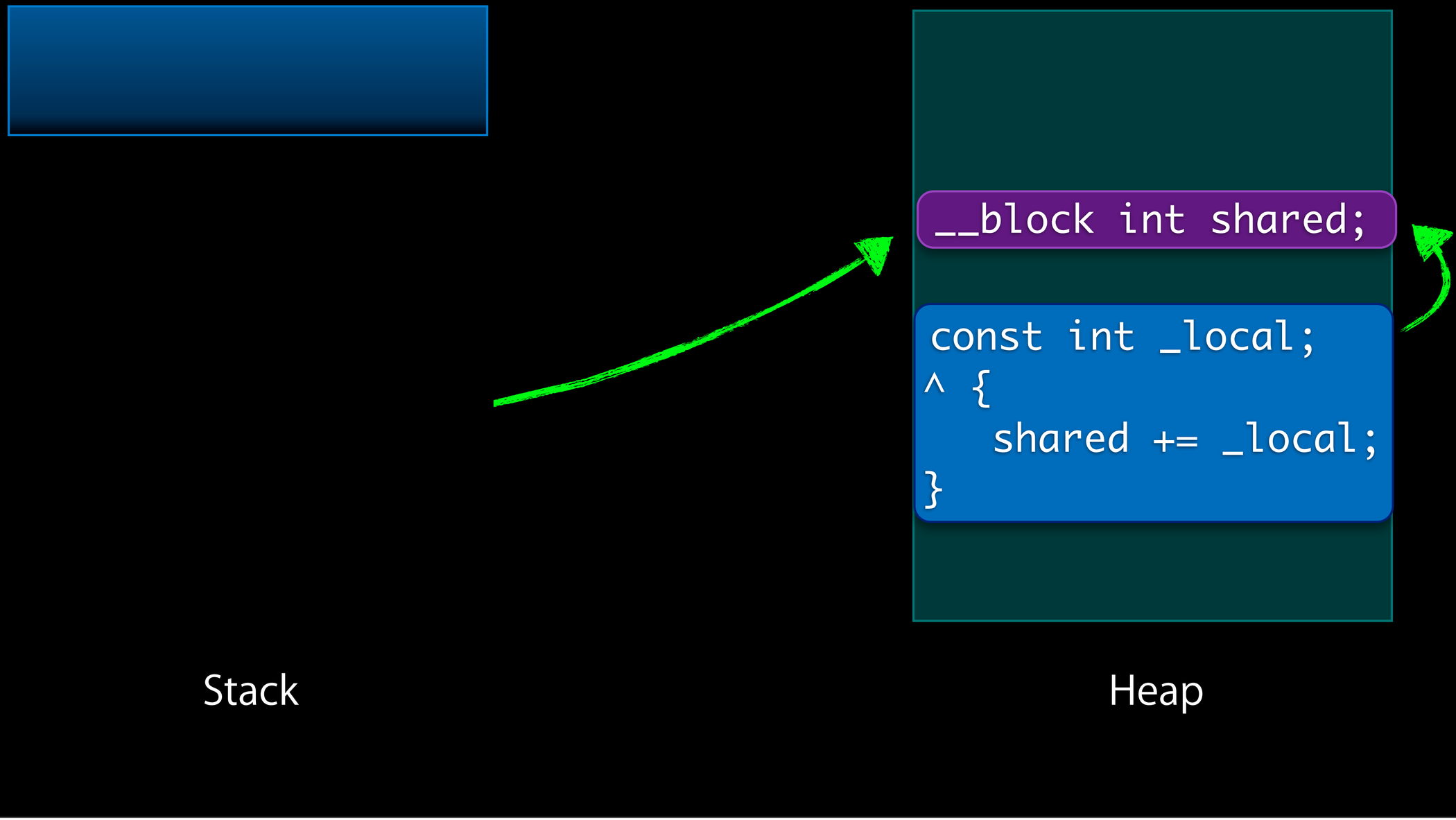
After Copy to Heap



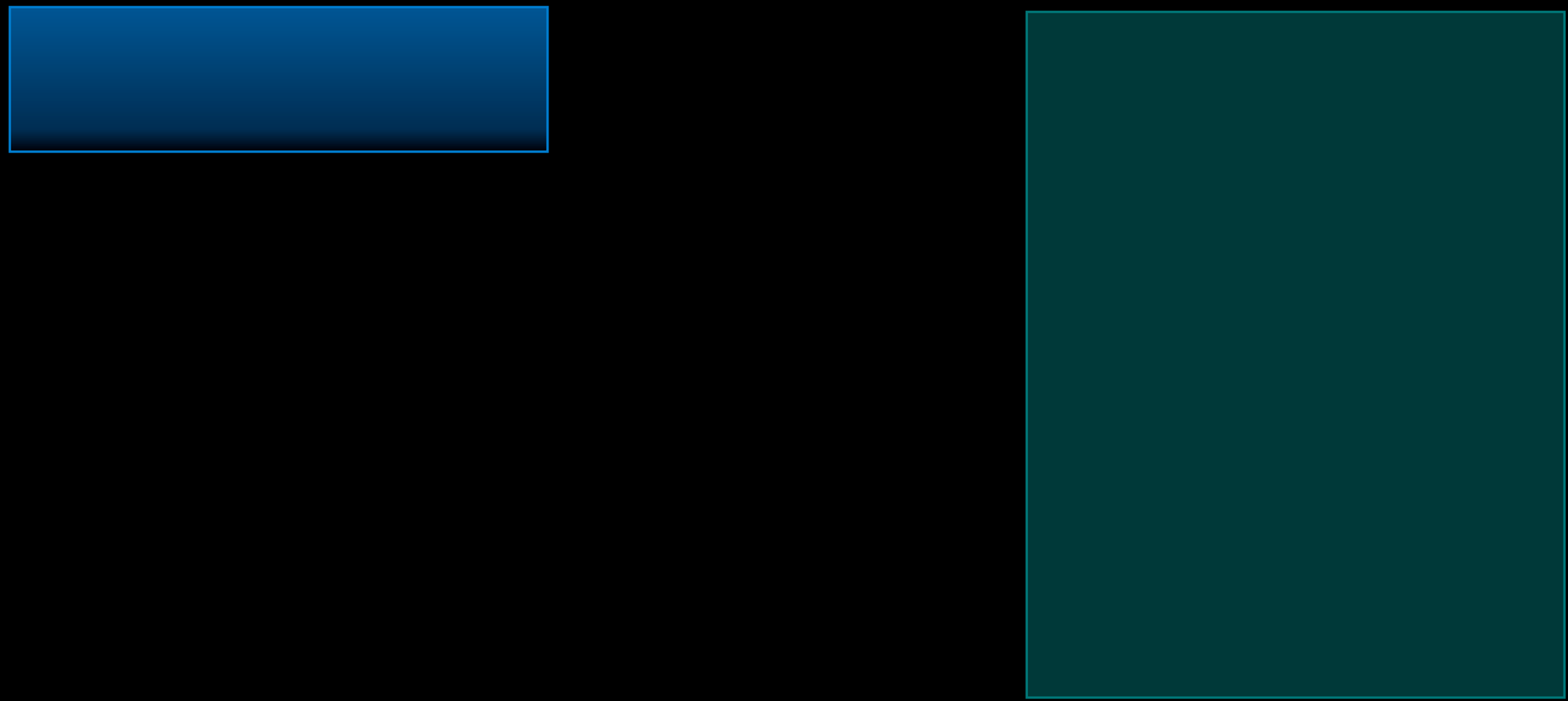
Case 1: Function Might Return…



Case 2: Heap Object Might Perish First



Leaving Us at Our Initial State



Stack

Heap

Objective-C Specializations

All Blocks Are Objects!

```
typedef void (^workBlk_t)(void);
```

```
- (void) keepThisBlock:(workBlk_t)aBlock {
   [myBlock release]; // skip if GC!
   myBlock = [aBlock copy];
 }
```

```
workBlk_t myBlock = [^{ printf("hi!\n") } copy];
```

```
@interface MyClass : NSObject

@property(copy) workBlk_t myBlock;

@end
```

The -copy method is much preferred over the Block_copy() primitive which must always be paired with Block_release()

Lifetime of Objects Used Within Blocks

```
- (void) yourMethod {
    id aLocal = ...;
    __block id blockVar = ...;
    ...
    ... = Block_copy( ^ {
        myIvar = [aLocal transform:blockVar];
    });
    ...
}
```

When copied:

1. aLocal will be retained
2. self will be retained
3. blockVar will be left alone

Surprises

A Block Reference Isn’t a Pointer

They can be cast to void *

Problem

```
printf("%p\n", ^{ ... });                    // complains
```

Solution

```
printf("%p\n", (void *)^{ ... });        // happy
```

Avoid Retaining Stack-Based Blocks!

They don’t complain but they also don’t copy

Problem

```
[aBlock release];
aBlock = [^{ ... } retain];
```

Solution

```
[aBlock release];
aBlock = [^{ ... } copy];
```

Pointers to Stack Structures

Out-of-scope structures

```
void someFunction() {
    MyStruct_t *array[10]);
    for (int i = 0; i < 10; ++i) {
        MyStruct_t wilma = { ... };
        array[i] = &wilma;
    }
    array[3]->...;                  // BOOM!!
}
```

References to Stack-Based Blocks

Out-of-scope Blocks

Problem

```
void someFunction() {
    void (^array[10])(void);
    for (int i = 0; i < 10; ++i) {
        array[i] = ^ { printf("captured i as: %d\n", i); };
    }
    ...
    array[3]();         // BOOM !!!
}
```

Solution

```
void someFunction() {
    void (^array[10])(void);
    for (int i = 0; i < 10; ++i) {
        array[i] = [^ { printf("captured i as: %d\n", i); }; copy];
    }
    ...
    array[3]();         // captured i as: 3
}
```

Unitialized Variables Can Be Copied

```
void doSomething(BOOL verbose) {
    NSString *boilerplate;

    if (verbose)
        boilerplate = ...;

    dispatch_async(NULL, ^ {
        ...;
        if (verbose)
            NSLog(@"%@: we got here!", boilerplate);
    });
}
```

Debugging

- gdb supports inspection of const captured variables
- gdb views __block variables properly
- Block construction stepped over properly in gdb
- Breakpoints can be set
- Instruments monitors GCD use of Blocks

Status

- Full support for Blocks in C & ObjC in both gcc 4.2 and clang
- Early support for C++, but fraught with can't-do
- Specification is published as part of clang open source
- Runtime is also now available as well

- Blocks was presented to the C1x standard committee
  - http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1370.pdf
  - Works for C unlike C++ lamba
  - Does much more than C++ lambda

More Information

Michael Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

Documentation

The Objective-C 2.0 Programming Language
Garbage Collection Programming Guide
Blocks Programming Topics

Related Sessions

Understanding Grand Central Dispatch in Depth	Presidio Thursday 9:00AM

Labs

Objective-C and Garbage Collection	OS X Lab C Wednesday 9:00AM
Objective-C and Garbage Collection	Snow Leopard Lab Friday 12:00 noon

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09